LO22 cours 8 (le dernier :-))

Partie 1 :

Rechercher quelque chose sous Linux (ou Unix...)

Que peut-on chercher ?

1) Un paquet, une application
2) De la documentation : comment fonctionne une application ? (syntaxe, options, limitations)
3) À exécuter une application
4) Un fichier
5) Quelque chose de précis : commande find
6) Une occurrence (chaîne de caractère, par exemple)

1 Rechercher...

1.1) Un paquet --> «package»

Problème : comment savoir que celui-ci existe ?

La méthode dépend de la distribution :

Debian : apt-cache/apt-get ; dselect ; synaptic

Mandrake : urpmi

Red Hat : rpm

Suse : yast/yast2

Exemple :

On recherche une application dont on a oublié le nom...(Pingouin qui descend les pistes de ski)

apt-cache search game

apt-cache search game 3D

apt-game search game 3D | grep race

Enfin :

apt-cache show tuxracer

Rechercher...

1.2) Une application :

Règle : La plupart du temps, l'application cherchée porte le même nom que le paquet. Mais ce n'est pas toujours vrai...

Exemples :

gqview (gqview pour l'application) ;

ImageMagick (--> display,convert...)

En utilisant la complétion :

--> on tape les deux premières lettres dans un terminal et on appuie sur la touche TAB

Le(s) choix possible(s) s'affiche, et on complète le nom jusqu'à avoir le nom complet. On ajoute ensuite les options/arguments voulus.

Rechercher...

2) De la documentation sur une application ou le Mode d'emploi d'une application

Une fois qu'on a le nom d'une application, on veut connaître son mode d'emploi :

man nom_application

OU

man numéro_de_section application

Après, il faut faire des essais, ou trouver des exemples...

3) Exécuter une application...

N.B. : on suppose que le fichier à «lancer» est aussi exécutable. Si l'application n'est pas exécutable, il faut utiliser chmod u+x...

A) En mode graphique

- Il faut que le **nom** de cette **application** apparaisse dans un **menu** (sinon, il faut créer une entrée).
- **Exécuter un raccourci** : combinaison de touches (définies par l'utilisateur), ou geste particulier avec la souris avec une application du type «Mouse Gestures»
- Ou encore exécuter la **boîte de dialogue** : «lancer l'application...», si elle est disponible.

B) Dans un terminal (donc en mode graphique...)

- Il suffit de taper le nom de l'application, qui doit être dans le PATH (**echo $PATH**), avec les options et éventuels arguments souhaités. Sinon, donner le chemin absolu ou relatif à l'endroit où l'on se trouve permettant de lancer l'exécution

C) Dans une console

- On ne peut lancer que des applications de type “texte”, sauf applications utilisant le frame-buffer (adressage direct de la mémoire vidéo).

Rechercher...

4) Un fichier (exécutable ou non) : locate, type et find

- **locate** : permet de trouver un fichier, présent dans une base de données, qui coïncide avec un motif de recherche
- Exemple :

locate module.conf

locate modules.conf

- **type** : donne le chemin d'accès d'un fichier exécutable.
- **eric@alube:~$ type openoffice**
- **openoffice is /usr/bin/openoffice**
- **eric@alube:~$**

5) Quelque chose de précis : commande **find**

Cette commande permet de retrouver des fichiers dans une hierarchie de répertoires.

Syntaxe : find [chemin] [expression]

[expression] est contituée d'options, de tests et d'actions.

NB. : Toutes les options renvoient toujours la valeur vraie. Elles ont toujours un effet global, plutôt que de s'appliquer uniquement à leur emplacement dans l'expression. Néanmoins, pour améliorer la lisibilité, il est préférable de les placer au début de l'expression.

- **Options possibles :**

-**daystart** : avec -atime, -ctime, mtime, -amin, cmin et -mmin

-**depth** : traite d'abord les sous-répertoires

-**follow** : permet de déréférencer les liens symboliques (comme si on accédait directement au lien pointé)

etc....

Tests possibles :

Argument numériques :

supériorité à une valeur : +n

inférieurité à une valeur : -n

égalité à une valeur : n

Exemples :

-amin n --> dernier accès au fichier il y a n minutes

-atime n --> dernier accès au fichier il y a n*24 heures.

Options de find (suite...)

-cmin n --> dernière modification du statut du fichier il y a n minutes.

-name motif --> Fichier dont le nom de base (sans les répertoires du chemin d'accès), correspond au motif du shell. Les méta-caractères (`*', `?', et `[]') ne sont jamais mis en correspondance avec un point `.' au début du nom. Pour ignorer un répertoire, ainsi que tous ses sous-répertoires, utiliser l'option -prune; vous trouverez un exemple dans la description de l'option -path.

-type k --> Fichier du type avec **k** à prendre parmi

- **b** fichier spécial en mode bloc (avec buffer)
- **c** fichier spécial en mode caractère (sans buffer)
- **d** répertoire
- **p** tube nommé (FIFO)
- **f** fichier régulier
- **l** liens symbolique
- **s** socket

etc...(voir man find)

Etude de quelques exemples :

-Donner la liste de ses propres fichiers

find ~/ -user `whoami` -print

-Donner la liste des fichiers non accédés depuis plus de 125 jours

find . -atime +125 -print

-Utilisation de la commande find pour supprimer tous les fichiers exécutables de nom a.out (résultat de gcc sans «-o nom_de_fichier_de_sortie», et des fichiers objet, qui n'ont pas été accédés depuis 5 jours.

find $HOME \(-name a.out -o -name "*.o" \) -atime +5 -exec rm {} \ ;

-Utilisation de la commande find, pour supprimer, dans le répertoire de travail, tous les fichiers dont le nom est core (type core-dump) ou se terminent par .o et qui correspondent à des fichiers dont la taille est supérieure à 3 blocs (de 512 octets)

find . \(-name core -o "*.o" \) -size +3 -exec rm {} \ ;

6) Une occurrence.

Exemple : recherche d'une chaîne de caractère

Commandes : grep, egrep

Exemples :

cat fichier | grep chaîne_recherchée

N.B. : UUOC = Useless Use Of Cat :-)

[mais cat augmente la lisibilité. Ne pas hésiter à l'utiliser quand même...]

grep chaîne_recherchée ./endroit

Exemple de recherche d'une occurrence dans un fichier : commande **egrep**

Exemple :

eric@tomate:/usr/src/linux-2.4.23pre5/crypto$ egrep -H generic_rotl32 ./*

./aes.c:u32 generic_rotl32 (const u32 x, const unsigned bits)

./aes.c:#define rotl generic_rotl32

eric@tomate:/usr/src/linux-2.4.23pre5/crypto$

Partie 2 :

Programmer sous d'autres shells

1) Shells existants

2) Changements de shells

3) Différences entre Bash2 et Bash1.14

4) Korn shell

5) Tcsh

6) Conclusion

1) Shells existants (les principaux :-))

- **sh --> GNU Bourne again shell**

```
eric@alube:~$ ls -l /bin/sh
lrwxr-xr-x    1 root     root            4 2003-11-28 09:54 /bin/sh -> bash
```

- **ash** (shell très léger, utilisé pour sa grande compacité)
- **bash** (Bourne Again sh) **-> par défaut sous Linux et Mac OS 10.3**
- **bsesh** (Scheme sh for BSE)
- **csh** (C sh)
- **tcsh** (tc sh) **--> par défaut sous Mac OS10.2**
- **wish** (windowing shell)
- **chroot** : pour la maintenance, principalement :-)

...man shell_choisi pour en savoir plus

2) Commandes permettant de changer de shell :

chsh [-s shell]

Ensuite le mot de passe utilisateur est demandé, et en cas de succès, la modification est effective dans /etc/password pour le prochain login

Remarques :

1) Il faut donner le chemin complet du shell...

2) Pour les changements, voir /etc/password

Vérification :

```
 eric@alube:~$ type tcsh | cut -d" " -f3
/usr/bin/tcsh
eric@alube:~$ chsh -s `type tcsh | cut -d" " -f3`
Password:
eric@alube:~$ cat /etc/passwd | grep eric | cut -d":" -f 5-10
alube,,,:/home/eric:/usr/bin/tcsh
eric@alube:~$ chsh -s `type bash | cut -d" " -f3`
Password:
eric@alube:~$ cat /etc/passwd | grep eric | cut -d":" -f 6-10
/home/eric:/bin/bash
```

3) Différences Bash2 / Bash1.14 (1ère version)

Amélioration apportées par Bash2 :

- les tableaux (inexistants sous Bash1.14)

- améliorations des fonctions internes et ajout de variables supplémentaires :

- quelques fonctions internes admettent des options supplémentaires dans Bash2. ulimit, time et shopt font leur apparition, certaines expressions deviennent obsolètes

--> en résumé, pas trop de problèmes, car Bash1 n'est plus très utilisé

4) Korn shell et pdksh

- Korn shell : à longtemps représenté l'équivalent commercial de Bash pour les Unix libres.
- La version libre s'appelle pdksh
- Interface plus agréable
- Puissant langage de programmation par scripts
- Bonne compatibilité entre Bash et Ksh (tous deux ont une bonne conformité vis à vis de la norme POSIX 1003.2 relative à la compatibilité entre Unix)

Quelques différences entre Korn shell et Bash :

- Les différences essentielles concernent l'utilisation interactive de l'interpréteur de commandes, les scripts peuvent presque toujours être interprétés directement.
- Le shell Korn ne permet pas d'utiliser simultanément le mot clé ***function*** et les doubles parenthèses. (c'est l'un ou l'autre)
- l'expresion obsolète $[expression] : interdite en Korn shell, tolérée en Bash
- Rappel : **$((expression))** est la syntaxe correcte
- **enable** n'existe pas en Korn shell
- **print** (équivalente à echo) existe en Korn shell, pas en Bash
- **declare** (bash) n'existe pas en Korn shell

Quelques différences Korn shell / Bash (suite) :

- Les signaux temps réels n'ont pas les mêmes noms sous Korn shell et sous Bash
- pdksh n'implémente pas les pseudo-signaux temps-réel DEBUG, ERR et EXIT.

En pratique : peu d'efforts sont nécessaires pour assurer la portabilité des scripts entre Korn shell et Bash (on évitera par exemple déclare dans les scripts en Bash si on a besoin de porter un script sous ksh)

Bonne portabilité en général entre Bash et Korn shell

Rappel : bien signaler le shell *avec le shebang :*

#! /bin/ksh

5) tcsh

- le tcsh est une implémentation améliorée (avec la complétion pour les noms de fichiers) du C shell de l'université de Berkeley --> ce serait une affaire de goût entre bash et tcsh.

Remarque : il existe un “consensus” pour ne plus programmer en C shell, à causes de bogues et d'incohérences dans l'implémentation du langage même

- gestion des variables totalement différente entre C shell et Bash :

Le C shell différencie les variables du shell des variables d'environnement (**set** et **setenv**)

- Les modificateurs des noms de variables sont complètement différents
- nombreuses incompatibilités entre tcsh et Bash quand à la gestion des signaux, des descripteurs de fichiers et des jobs...etc.

Conclusion : A éviter si on utilise aussi Bash

6) Conclusion

- Il existe une grande quantité de shells disponibles sous Linux...
- Exemple : **ash (créé par Kenneth Almquist)** , qui permet de démarrer sur disquette, est un shell très compact, même compilé en statique (même taille en compilé en statique (autonome) que les autres en compilé en dynamique (linkés à la libC)
- **zshell** : solution de remplacement/concurrent annoncé de bash ?
- En effet, zsh : allie une interactivité très agréable à des performances comparables aux autre grands shells.

Bibliographie :

1) **Langages de script sous Linux**

Christophe BLAESS

Editions Eyrolles 2002

2) **Le Système Linux (2ème édition 98)**

Matt Welsh, Matthias Kalle Dalheimer & Lar Kaufman

Editions O'Reilly 1998

Bibliographie (suite)

3) **Linux in a Nutshell**

Jessica P. Hekman

Editions O'Reilly 1997

4) **Unix : le tout en poche**

Dave Taylor & James C. Armstrong Jr

Editions Simon and Shuster MacMillan 1998

FIN du cours initiation à Unix...